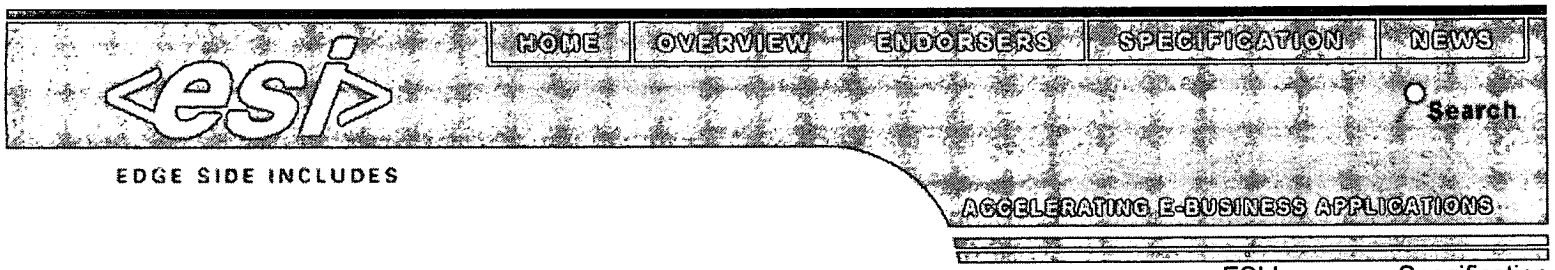


ESI Language Specification
Edge Architecture Specification
ESI Invalidation Protocol
JESI Tag Library Specification

Technical Specification

JESI Tag Library 1.0 Specification: Tags for Edge-Side Includes in JSP

Contributors:
Julie Basu, Ajay Desai, Ping Guo, Larry Jacobs, Xiang Liu, Cyril Scott, Mark Tsimelzon, Brian Watson, Alex Yiu
Version: 1.0
Status : First draft

Table of Contents:

G als

The purpose of this document is to describe the syntax and semantics of the JSP tags in the JESI 1.0 tag library.

Audience of this Document

We assume the readers of this document have the basic understanding of ESI Specification 1.0 and JavaServer Pages application and JavaServer Pages Tag Library.

Overview

JESI tags Simplify the following tasks associated with ESI use from JSPs

- **Template declaration** - Page template and fragment definition
- **Content invalidation** - Invalidation message generation to remove Page template and fragments from ESI processors
- **Page personalization** - tagging pages with personalized content which can be executed in an ESI processor

Motivation for JESI Tags

The JESI tag library is intended to be the convenient JSP application-level interface to the ESI tags for web caching functionality. Its purpose is to facilitate the use of ESI tags within a JSP application.

While ESI tags can be directly used in JSPs and servlets, there are several reasons why a custom JSP tag library on top of ESI is useful for JSP developers:

- **Standard JSP framework:** Although ESI is XML-based and looks similar to a custom JSP tag library, it is nevertheless a different kind of mark-up language. JSP developers and JSP IDE tools are used to dealing with the standard framework of JSP custom tag libraries. For example, the path references in a JSP page are either page-relative or application-relative. These URIs can be translated by the JESI tag library into absolute path references for ESI.
- **Short-cut syntax:** The JESI tag library can simplify common tasks such as:
  - Specifying meta-data information, such as expiry time of page fragments, conveniently within the JESI tags.
  - Sending invalidation messages to purge URLs
  - Personalizing dynamic pages using cookie information

  The JESI tags can translate these tasks into appropriate calls, such as generating a HTTP request for invalidation, translating into the appropriate ESI tag in the generated page, setting the appropriate HTTP response header etc.
- **Convenient application-level configuration files:** The JESI tag library can make use of application-level configuration files to specify deployment-time parameters and default settings for various options. The config file allows parameters such as the username and password for invalidation to be conveniently specified externally at deployment time, without requiring changes to the application code.
  It can also simplify the JSP code that must be written to utilize ESI functionality by specifying application-specific defaults.

How Does JESI W rk?

The JESI tag library will be installed on the customer's application server. It could be either be pre-installed by the application server vendor, or be installed directly by the customer.

JESI is layered on top of the ESI framework. No changes to the ESI language syntax, semantics, or implementation are necessary.

Instead of using ESI tags such as <esi:include> directly, the JSP developer will be using the JESI tags such as <jesi:include>. The Java code implementing the library will be translating the JESI tags into corresponding ESI tags. Note that the translation may not always be a simple one-to-one mapping.

There will be a tag library jar file containing the JESI tag handler implementation and a TLD file that describes the JESI tags and attributes.

JESI 1.0 Proposal

There are two parts to the JESI 1.0 specification:

- Mandatory JESI tags, which must be implemented by a JESI 1.0-compliant tag library
  - for a mandatory tag, there may be some implementation optional attributes. Some implementation may choose not to implement in this version. Please note that this concept is different from the usage optional, where developers choose to use a particular feature or not. Implementation-optional does imply usage-optional, but not vice versa. If an attribute is marked as just "optional" within this document without explicitly specifing "usage" or "implementation", then it means the attribute is usage-optional.
- Optional JESI tags, which *may* be implemented by a JESI 1.0-compliant tag library.

Template declaration

Usage Models

The JESI 1.0 specification supports two different models of page fragment caching, for

- JESI-enabling new JSP pages
- JESI-enabling existing JSP pages.

JESI-enabling New Web Pages

This model is recommended for applications that are being newly written. The JSP developer factors a page into independent JSP fragments, and indicates how these JSP fragments are to be assembled into an "aggregate" JSP.

In this model, there are two tags:

- <jesi:include> - for declaring in templates where fragments shall be included in the ESI processor; this JSP tag generates `<esi:include>` tag (refer to [2]).
- <jesi:control> - for declaring the attributes of templates and fragments; e.g., expiration; this JSP tag corresponding to the control-headers of edge architecture in [1]. This tag is implementation-optional. The usage of this tag is optional. Without this tag, default control-header properties will be still set in the HTTP headers

JESI-enabling Existing Web Pages

This model is recommended for applications that already exist and can be assembled from an ESI processor. In this model an existing JSP is enhanced to identify the page's template and fragments.

In this model the following tags are used for demarcating the page

- <jesi:template> - for declaring a template and its attributes: e.g. expiration
- <jesi:fragment> - for declaring a fragment within a template
- <jesi:include> - same as above
- <jesi:param> - used within a <jesi:include> tag to specify additional parameters; this tag is implementation optional
- <jesi:codeblock> - for declaring a code block to be executed during template generation ; this tag is implementation optional

Before providing the detailed syntax and semantics of these tags, we illustrate their use through some examples.

Example 1: Independent Page Fragments

Assume that a page `welcome.jsp` is made up of three independent fragments:

- `stocks.jsp`, which provides financial information on stocks
- `weather.jsp`, which shows the weather conditions
- `sales.jsp`, which lists the special sales events of interest to the user.

The aggregate portal page `welcome.jsp` may be assembled from these three JSP fragments using just the <jesi:include> tag as follows:

```
<html>
 <body>
   <jesi:include page="stocks.jsp" flush="true" />
 <p>
 <hr>
   <jesi:include page="/weather.jsp" flush="true" />
 <p>
 <hr>
  <jesi:include page="../sales.jsp" flush="true" />
 </body>
</html>
```

In this example, the expiration of both portal template page and individual fragments follows the default setting. Or, developers can optionally put <jesi:control> within the including template JSP or individual fragment JSP, such as, stocks.jsp. When this page (say, the URL of this page is "`http://host:port/application1/dir2/dir3/welcome.jsp`" is requested, the following template will be generated from Java layer:

```
<html>
 <body>
   <esi:include src="http://host:port/application1/dir2/dir3/stocks.jsp" />
 <p>
 <hr>
   <esi:include src="http://host:port/application1/weather.jsp" />
 <p>
 <hr>
  <esi:include src="http://host:port/application1/dir2/sales.jsp" />
 </body>
</html>
```

Based on this template, ESI processor will fill in the included fragment content from its cache. It will request individual fragments if the cache copy does not exist or is out of date.

Example 2: Page Fragments with Varying Template

In some cases, the layout of the aggregate page may not be fixed. It may pick up different fragments depending on the user profile or other request parameters. In this e-commerce website example, a cookie is used to represent the identity of a customer. If the cookie does not present, this URL will show a generic welcome page with general product recommendation. However, if the cookie presents, the server logic will retrieve a list of recommended products and their page URLs based on the customer profiling. The list of recommended products URL will be listed in the template by `<jesi:include>` which in turn generates `<esi:include>` for any downstream ESI processor to process.

```
<%
  String customerId=CookieUtil.getCookieValue(request,"customerid");
  if (customerId==null) {
    // some unknown customer
%>
    <jesi:include page="genericwelcome.jsp" />
<%
  }
  else {
    // a known customer; trying to retrieve recommended products from profiling
    String recommendedProductsDescPages[]=ProfileUtil.getRecommendedProductsDescURL(customerId);
    for (int i=0; i<recommendedProductsDescPages.length; i++) {
%>
    <jesi:include page="<%=recommendedProductsDescPages[i]%>" />
<%
    }
  }
%>
```

Example 3: Page Fragments with Parameters Here is the source of the main.jsp:

```
<html>
<jesi:control cache="no" />
<jesi:include page="a.jsp?p1=v1" />
<h3>hello ...</h3>
<jesi:include page="b.jsp" />
```

```
<h3>world ...</h3>
<jesi:include page="c.jsp?p1=v2" copyparam="true" />
</html>
```

In this example, the main.jsp has three fragments: a.jsp, b.jsp and c.jsp. c.jsp functions based on another HTTP parameter "p2" passed from main.jsp. This is taken care by setting the "copyparam" attribute to true. When main.jsp is requested through "http://host:port/application1/main.jsp?p2=abc", the following ESI tag will be generated for `<jesi:include>` of c.jsp:

```
<esi:include src="http://host:port/application1/c.jsp?p1=v2&p2=abc" />
```

Please note that copyparam should be used only in a non-cacheable template (see above `<jesi:control>` tag). Because, the actual src attribute will change from one request to another.

Example 4: Using <jesi:template> and <jesi:fragment>

Here is a main.jsp:

main.jsp

```
<jesi:template expiration="3600">
HTML #1
<% // some Java code got always executed Java-code-#1 %>
<jesi:fragment expiration="60">
JSP code block #1
</jesi:fragment>
HTML #2
<jesi:fragment>
JSP code block #2 using default expiration
</jesi:fragment>
HTML #3
<jesi:fragment expiration="600">
JSP code block #3
</jesi:fragment>
HTML #4
</jesi:template>
```

Content invalidation

The JESI tag library provides the following tag:

- <jesi:invalidate>

for explicitly removing and/or expiring selected objects cached in an ESI processor.

Cached objects may need to be invalidated depending on external circumstances such as changes to the underlying data in SQL tables. It is also possible that the execution of one object (or page) invalidates other cached objects.

Page Personalization

A common requirement of web applications is the inclusion of user-specific information in a web page. For example, a welcome page for an application may contain the user's name in a greeting message. This makes the web page necessarily dynamic and dependent on cookie or session information. Without special tags to indicate this dependency to the ESI processor, the web page cannot be shared by multiple user sessions.

The JESI 1.0 Tag Library

Template declaration

The <jesi:template> tag

ESI template declaration tag

Syntax

```
<jesi:template
     [expiration="value"]
     [maxRemovalDelay="value"]
     [cache={"yes", "no", "no-remote"}]
     [control="control_str"]
     [useRelativeURLs="true" | "false" ]
     [scheme="scheme_str"]
     [serverName="server_name_str"]
     [serverPort="server_port_str"]
     [servletPath="servlet_path_str"]
     [pathInfo="path_info_str"]
```

```
[executeEntirePage="true" | "false"]
   >
```

Description

The `template` tag is used to declare an ESI template within an existing JSP.

The `template` tag must appear before all the JESI tags or any buffer flush, while the end tag must appear after all other JESI tags. If this tag happens more than once within one JSP, a `JspTagException` will be thrown.

The attributes of the template tag are used to specify the property of the template itself. e.g. whether the template itself is cacheable, and if so for how long. The attributes are the same for the `<jesi:control>` tag. This `template` tag will set corresponding HTTP headers according to Edge Architecture Specification [1].

`expiration` an optional parameter, is the template's lifetime in seconds from insertion in the ESI processor. The default is never-expire.

`maxRemovalDelay` an optional parameter, is the maximum time, in seconds, after the template's expiration that the ESI processor may store the template.The default is zero, that means immediate removal.

`cache` an optional attribute to specify whether the template is cachable: yes, no, or no-remote. The default is yes.

`control` an optional attribute that allows users to specify the ESI control headers directly, not just expiration and cacheability

`useRelativeURLs` If this attribute is set to true, all generated `<esi:include>` tags will have relative URLs as the src attribute value. Otherwise, all generated URLs will be absolute. The default value of useRelativeURLs is false.

`scheme` This attribute sets the protocol specified in all URLs generated within the body of the `<jesi:template>` tag. If this attribute is not set, then the value of request.getScheme() is used. If an expression is used to set the attribute value, that expression must result in a string. NOTE: In the case of `<jesi:include>` tags, this value will only affect URLs that do not specify a protocol.

`serverName` This attribute sets the domain specified in all URLs generated within the body of the `<jesi:template>` tag. If this attribute is not set, then the value of request.`getServerName()` is used. If an expression is used to set the attribute value, that expression must result in a string. NOTE: In the case of `<jesi:include>` tags, this value will only affect relative URLs.

`serverPort` This attribute sets the port specified in all URLs generated within the body of the `<jesi:template>` tag. If the value of port ids the default port for the specified protocol, the port will not be explicitly set in the generated URLs. If this attribute is not set, then the value of request.`getServerPort()` is used. If an expression is used to set the attribute value, that expression must result in a string of a primitive integer.

`servletPath` an optional attribute that allows users to specify the servlet path explicitly; it is extra useful, when the request is under forwarding; it can be used to specify servlet path of the forwarding page.

`pathInfo` an optional attribute that allows users to specify the path info explicitly; it is extra useful, when the request is under forwarding; it can be used to specify path info of the forwarding page.

`executeEntirePage` an implementation optional boolean attribute, the default is false; if true, the whole page including the template and all the fragments will be executed.

Attribute Names	Implementation Optional	Usage Optional	Can be Request Time Attribute
expiration	yes	yes	yes
maxRemovalDelay	yes	yes	yes
cache	yes	yes	yes
control	no	yes	yes
useRelativeURLs	yes	yes	no
scheme	yes	yes	yes
serverPort	yes	yes	yes
servletPath	no	yes	yes
pathInfo	no	yes	yes
executeEntirePage	yes	yes	yes

The <jesi:include> tag

The include tag is used to identify JSPs for generating page fragments.

Syntax

```
<jesi:include
    page= "url_str" | "<%= expression %>"
    [flush="true"|"false"]
    [copyparam="true"|"false"]
    [alt="alt_page_str"]
    [ignoreError="true"|"false"]
       />
```

Or

```
<jesi:include
    page="url_str" | "<%=expression %>"
    [flush="true"|"false"]
    [copyparam="true"|"false"]
    [alt="alt_page_str"]
    [ignoreError="true"|"false"]
       >
    ( <jesi:param
        name="paramterName"
        value="{ parameterValue | <%= expression %> }"
        /> )+
</jesi:include>
```

Description

The `<jesi:include>` tag identifies a JSP page for generating the desired page fragment. The tag is empty, except for an optional form where it includes parameters extracted from JSP include calls.

The main function of this tag is to generate <esi:include> tag embedded in the resulting template. It will generate only "src" attribute of the <esi:include> tag. It will not generate the "alt" and "onerror" attributes. It will generate the "src" attribute based on the given "page" attribute, "copyparam" and any enclosed <jesi:param> tags. The attributes of this tag resemble <jsp:include> to ease the migration from <jsp:include> to <jesi:include>. <jesi:include> is NOT intended to a perfect substitute of <jsp:include>. There are cases of which <jsp:include> cannot be substituted by <jesi:include>. For example, the including main page insert a Java object attribute into the HTTP servlet request object and passed to the included page.

page is a required attribute of the JSP page fragment, and can be specified either in the page-relative or application-relative format. It has the extra application root / context path processing. So, JSP developers do not need to hard code the deployment application path in this tag. (e.g. "http://host:port/publicapp/dir1/a.jsp" vs "http://host:port/betaapp/dir1/a.jsp".

flush is an optional attribute which is always ignored; the existence of this attribute make it easier for the existing `<jsp:include>` users to switch to `<jesi:include>`.

copyparam is an implementation-optional and usage-optional attribute; if the value is true, the "src" attribute of generated ESI tag will be appended by the parameter string of the original HTTP request (not including HTTP posted parameter body). This attribute can be used only in a non-cacheable template page, since the original request parameters can change in each HTTP request. It should throw an JspTagException exception, otherwise. The default is false.

`alt` is an implementation-optional attribute that is used to generate the "alt" attribute in `<esi:include>`. The prepending logic for application path will be applied to this attribute, similar to the "page" attribute of this tag, if this "alt" attribute does not contain "http://" or "https://".

`ignoreError` is an implementation-optional boolean attribute. The default is false. If the attribute is true, under error conditions during ESI include, the ESI processor will continue with the rest of hte page. If the attribute is false, under error conditions during ESI include, the ESI processor raise an ESI exception.

Attribute Names	Implementation Optional	Usage Optional	Can be Request Time Attribute
page	no	no	yes
flush	no	yes	yes
copyparam	yes	yes	no
alt	yes	yes	yes
ignoreError	yes	yes	yes

`<jesi:param>` is a tag enclosed inside the `<jesi:include>` tag; the `<jesi:include>` tag will insert the parameter name and parameter value as a form of HTTP query string after the given page attribute and before the query string from the original request. The insertion order is same as the `<jesi:param>` tag order. the <jesi:param> tag itself is ***implementation optional***; however, the attributes are not implementation optional, once decided to be implemented.

Attribute Names	Implementation Optional	Usage Optional	Can be Request Time Attribute
name	no	no	no
value	no	no	yes

The <jesi:fragment> tag

Syntax

```
<jesi:fragment
     [expiration="value"]
     [maxRemovalDelay="value"]
     [cache="yes" | "no" | "no-remote"]
     [control="control_str"]
        >
```

JSP code fragment

```
</jesi:fragment>
```

Description

The `<jesi:fragment>` tag is used to demarcate a JSP code fragment within an existing JSP. When a specific fragment is requested (by specifying the esifrag query parameter in the request URL), only the content for that fragment is returned. This tag must be declared within the body of a <jesi:template> tag. This version of jesi:fragment does not support direct nesting within one page.

`expiration` an optional (both usage and implementation) parameter, is the template's lifetime in seconds from insertion in the ESI processor. The default is never-expire.

`maxRemovalDelay` an optional (both usage and implementation) parameter, is the maximum time, in seconds, after the template's expiration that the ESI processor may store the template.The default is zero, that means immediate removal.

`cache` an optional (both usage and implementation) attribute to specify whether the template is cachable: yes, no, or no-remote. The default is yes.

`control` a string attribute that sets the metadata for the fragment content. This value is set as the value of the ESI-control response header field. The header is set only if the specific fragment's content is requested, not if the template content is requested.

Attribute Names	Implementation Optional	Usage Optional	Can be Request Time Attribute
expiration	yes	yes	yes
maxRemovalDelay	yes	yes	yes
cache	yes	yes	yes
control	no	yes	yes

The <jesi:codeblock> Tag

The <jesi:codeblock> tag controls the execution of ESI request dependant JSP code. This tag is implementation optional.

Syntax:

```
<jesi:codeblock execute="template" | "fragment" | "always" >
...Request Dependant JSP Content...
</jesi:codeblock>
```

Description:

Often, a JSP page will have some piece of code that needs to be executed before any other fragment is executed (a database connection needs to be established, user id computed, etc...). Sometimes the same piece of code also needs to be executed for the template, and sometimes it does not. The <jesi:codeblock> tag provides this level of control over the execution of the block. When implemented, this tag allows a user to control what type
of request will cause the body of the tag to be executed. This tag must be declared within the body of a <jesi:template> tag.
NOTE: It is assumed that the code within the body of the <jesi:codeblock> tag does not produce content that must be returned to the requestor. Any user visible content produced within the body of a <jesi:codeblock> tag is suppressed.

Attributes
`execute` The type of request that will cause the body of this tag to be executed. The following values have the specified behavior:

- template: Only execute the tag body if the entire template is requested. If a specific fragment is requested, the body is not executed.
- fragment: Only execute the tag body if one of the fragments was requested. If the entire template is requested, the body is not executed.
- always: Always execute the tag body, regardless of the request type. This is the same behavior as not specifying the tag at all

The tag itself is implementation optional. However, the execute attribute is not, once decided to implement this tag.

Attribute Names	Implementation Optional	Usage Optional	Can be Request Time Attribute
execute	no	no	no

The <jesi:control> Tag

The attributes of the control tag are used to specify ESI processor behavior of the object (template or fragment) e.g. whether the object is cacheable, and if so for how long. The attributes are the same for the `<jesi:template>` tag. This tag is implementation optional.

Syntax

```
<jesi:control
     [expiration="value"]
     [maxRemovalDelay="value"]
     [cache= "yes" | "no" | "no-remote" ]
     [control="control_str"]
     [useRelativeURLs= "true" | "false" ]
     [scheme="scheme_str" ]
     [serverName="server_name_str"]
     [serverPort="server_port_str"]
     [servletPath="servlet_path_str"]
     [pathInfo="path_info_str"]
        />
```

Description

The `control` is an empty tag used to define how an ESI processor shall handle the object. It may be used only once in any given JSP, and shall not be used in JSPs with a <jesi:template> tag. The `control` tag will set corresponding HTTP headers according to Edge Architecture Specification [1].

The `control` tag must appear before any other JESI tags and buffer flush. If the `control` tag happens more than once within one JSP, a `JspTagException` will be thrown.

`expiration` `an optional attribute,` is the object's lifetime in seconds from insertion in the ESI processor. The default is never-expire.

`maxRemovalDelay` `an optional attribute,` is the maximum time, in seconds, after the object's expiration that the ESI processor may store the object. The default is zero, that means immediate removal.

`cache` an optional attribute to specify whether the object is cachable: yes, no, or no-remote. The default is yes.

`control` an optional attribute that allows users to specify the ESI control headers directly, not just expiration and cacheability

`useRelativeURLs` If this attribute is set to true, all generated `<esi:include>` tags will have relative URLs as the src attribute value. Otherwise, all generated URLs will be absolute. The default value of useRelativeURLs is false.

`scheme` This attribute sets the protocol specified in all URLs generated within the body of the `<jesi:template>` tag. If this attribute is not set, then the value of request.getScheme() is used. If an expression is used to set the attribute value, that expression must result in a string. NOTE: In the case of `<jesi:include>` tags, this value will only affect URLs that do not

specify a protocol.

serverName This attribute sets the domain specified in all URLs generated within the body of the `<jesi:template>` tag. If this attribute is not set, then the value of request.`getServerName()` is used. If an expression is used to set the attribute value, that expression must result in a string. NOTE: In the case of `<jesi:include>` tags, this value will only affect relative URLs.

serverPort This attribute sets the port specified in all URLs generated within the body of the `<jesi:template>` tag. If the value of port ids the default port for the specified protocol, the port will not be explicitly set in the generated URLs. If this attribute is not set, then the value of request.`getServerPort()` is used. If an expression is used to set the attribute value, that expression must result in a string of a primitive integer.

servletPath an optional attribute that allows users to specify the servlet path explicitly; it is extra useful, when the request is under forwarding; it can be used to specify servlet path of the forwarding page.

pathInfo an optional attribute that allows users to specify the path info explicitly; it is extra useful, when the request is under forwarding; it can be used to specify path info of the forwarding page.

Attribute Names	Implementation Optional	Usage Optional	Can be Request Time Attribute
expiration	yes	yes	yes
maxRemovalDelay	yes	yes	yes
cache	yes	yes	yes
control	no	yes	yes
useRelativeURLs	yes	yes	no
scheme	yes	yes	yes
serverPort	yes	yes	yes
servletPath	no	yes	yes
pathInfo	no	yes	yes

Content Invalidation

The <jesi:invalidate> tag

The `<jesi:invalidate>` tag provides the JSP programmer with an explicit way to remove and/or expire a specific object, or

objects, cached in an ESI processor. This is an implementation optional tag.

Syntax
```
<jesi:invalidate
    [url="url" username="value"password="value"]
   <jesi:object uri="..."/>
</jesi:invalidate>
```

Or
```
<jesi:invalidate
    [url="url" username="value" password="value"]
   <jesi:object uri="..." [prefix="yes"] [maxRemovalDelay="..."]>
      [<jesi:cookie name="..." value="..."/>]
      [<jesi:header name="..." value="..."/>]
   </object>
</jesi:invalidate>
```

Description

In its simplest form, the `<jesi:invalidate>` tag is used to select a cached object based upon its (HTTP) URI. In another form, all cached objects that share the same URI prefix may be selected. In its general form, the tag selects an object based upon the conjunctive combinationof the following:

- URI or URI prefix
- A cookie name-value pair
- A HTTP/1.1 request header

`url`, **`username`**, **`password`**, the URL, login username and login password for the cache server. If omitted, the values must be specified in the JESI configuration file.
`uri`, the full (or prefix) URI of the page(s) to be invalidated.
`prefix`, qualifies **`uri`**: if "yes" its a prefix URI,else (the default) its a full URI.
`maxRemovalDelay`, the maximum period, in seconds, that the invalidated page(s) can be used for a cache hit. The default value is zero.
`cookie`, optional cookie value.
`header`, optional header value.

Examples

1. Invalidate a single object in the default ESI processor:

```
<jesi:invalidate
    <jesi:object uri="/images/logo.gif"/>
</jesi:invalidate>
```

2. Invalidate all objects in the default ESI processor:

```
<jesi:invalidate
    <jesi:object uri="/" prefix="yes"/>
</jesi:invalidate>
```

3. Invalidate a single object but allow it to be served stale for upto 30 mins:

```
<jesi:invalidate
    <jesi:object uri="/images/logo.gif" maxRemovalDelay="1800"/>
</jesi:invalidate>
```

4. Invalidate a multi-version object:

```
<jesi:invalidate
    <jesi:object uri="/page.htm">
      <jesi:cookie name="user_type" value="Customer"/>
    </object>
</jesi:invalidate>
```

Page Personalization

The <jesi:personalization> Tag

This tag is used to easily enable a JSP for personalization in an ESI processor. For example, cookie value replacement in ESI processor, not in the original web server. It achieves this functionality by generating `<esi:vars>` tag. This tag is implementation optional.

Syntax

```
<jesi:personalize name="a_name_str" value="a_value_str" />
```

name a required attribute. It specifies the name of the cookie used to personalize the page by cookie value replacement.
value an usage-optional attribute. If it is not present, then no default value is generated in the <esi:vars> tag.

Example

```
<jesi:personalize name="user_id" value="Guest" />
```
This JESI tag will generate an ESI tag similar to this:

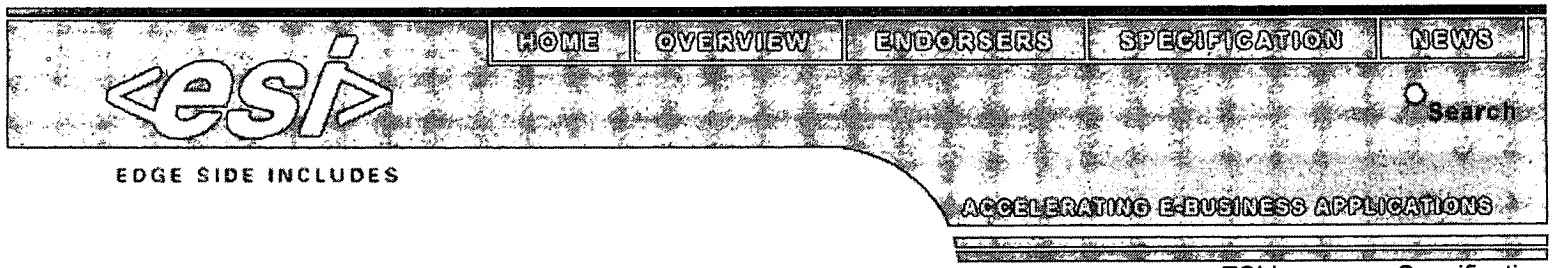
```
<esi:vars> $(HTTP_COOKIE{"user_id"}|"guest") </esi:vars>
```

which means ESI processor will try to retrieve the value of "user_id" cookie and do a value replacement in the ESI processor without sending a request back to the original server. If the user_id cookies cannot be found, the ESI processor may use the default value "guest" or go back to the original web server to get the cookie.

back to top

***

Home | Overview | Endorsers | Specification | News | Contact Us



ESI Language Specification
Edge Architecture Specification
ESI Invalidation Protocol
JESI Tag Library Specification

Technical Specification

ESI Invalidation Protocol 1.0

Authors:
Larry Jacobs, Oracle
Gary Ling, Oracle
Xiang Liu, Oracle

Table of Contents:

Abstract

This specification defines the ESI Invalidation Protocol, to allow for tight coherence between origin serves and surrogates (also know as "Reverse Proxies").

Introducti n

This document describes invalidation protocols for doing ESI invalidation of cached page documents. An invalidation request is an XML document sent over HTTP using HTTP/1.1 POST method. The syntax for an invalidation request is dictated by WCSinvalidation.dtd, the DTD (Document Type Definition) file which is defined later.

There is no official HTTP port reserved for invalidation yet, but there should be effort to petition for an official invalidation HTTP port. An implementation uses port 4001 for example.

Invalidation request authentication is done by using HTTP/1.1 basic authentication. An invalidation account should be created for this purpose. In one implementation, all invalidation is done by using the identity of a special account called "invalidator".

Invalidation request. The body of the invalidation is a valid XML document, in which a list of one or more invalidation objects is given. An invalidation object consists of a *selector-action* pair which means for all page documents that match the *selector*, *action* should be applied. An invalidation object completes successfully if the action is successfully applied to all the documents selected. An entire invalidation request is successful if all of its invalidation objects complete successfully.

If no page document is selected in an invalidation object, the invalidation object is still considered successful in the sense that it is over an empty set of page documents.

Invalidation response. Invalidation response is sent over HTTP as well. If the invalidation request is successful, the HTTP response code is `200`, and the body of the invalidation response is a valid XML document, in which a list of invalidation object results is given-one for each corresponding invalidation object in the invalidation request. An invalidation object result consists of a *selector-result* pair, in which *selector* is the same as that of the corresponding invalidation object and *result* details the status of invalidation for this invalidation object.

If the invalidation fails, a non-`200` HTTP response code is returned. And the body of invalidation response is just an HTTP message, describing further error message.

Invalidation DTD for Requests and Responses

One of the goals of designing the invalidation DTD is to make it as general as possible so that it can accommodate wide range of requirements and future development.

A valid invalidation message has to be a valid XML instance in the first place. Therefore, the first line of the document has to be

```
<?xml version="1.0"?>
```

Note that no white space is allowed before "<".

Through the document, any literal use of ampersand "&", less-than "<", greater-than ">", double-quote"", and apostrophe "'" has to be escaped with "&", "<", ">", """ and "'" respectively. For more information, please refer to XML standards.

The following shows the entire content of `WCSinvalidation.dtd`.

```
<!-- WCSinvalidation.dtd is the DTD file that describes a valid

-- Webcache invalidation XML message for request and response.

-->


<!-- root element for invalidation request -->

<!ELEMENT INVALIDATION (SYSTEM?,OBJECT+)>


<!-- VERSION is currently "WCS-1.0" without the quotes -->

<!ATTLIST INVALIDATION

VERSION CDATA #REQUIRED

>


<!ELEMENT SYSTEM (SYSTEMINFO+)>


<!ELEMENT SYSTEMINFO EMPTY>

<!ATTLIST SYSTEMINFO

NAME CDATA #REQUIRED

VALUE CDATA #IMPLIED

>
```

```
<!ELEMENT OBJECT ((BASICSELECTOR|ADVANCEDSELECTOR), ACTION)>

<!ELEMENT BASICSELECTOR EMPTY>

<!ATTLIST BASICSELECTOR

URI CDATA #REQUIRED

>

<!ELEMENT ADVANCEDSELECTOR (COOKIE|HEADER|OTHER)*>

<!ATTLIST ADVANCEDSELECTOR

URIPREFIX CDATA #REQUIRED

HOST CDATA #IMPLIED

URIEXP CDATA #IMPLIED

METHOD CDATA #IMPLIED

BODYEXP CDATA #IMPLIED

>

<!ELEMENT COOKIE EMPTY>

<!ATTLIST COOKIE

NAME CDATA #REQUIRED

VALUE CDATA #IMPLIED

>
```

```
<!ELEMENT HEADER EMPTY>
<!ATTLIST HEADER
NAME CDATA #REQUIRED
VALUE CDATA #IMPLIED
>

<!ELEMENT OTHER EMPTY>
<!ATTLIST OTHER
TYPE CDATA #REQUIRED
NAME CDATA #REQUIRED
VALUE CDATA #IMPLIED
>

<!ELEMENT ACTION EMPTY>
<!ATTLIST ACTION
REMOVALTTL CDATA #IMPLIED
>
```

```
<!-- root element for invalidation response -->

<!ELEMENT INVALIDATIONRESULT (SYSTEM?, OBJECTRESULT+)>


<!-- VERSION is currently "WCS-1.0" without the quotes -->

<!ATTLIST INVALIDATION

VERSION CDATA #REQUIRED

>

<!ELEMENT OBJECTRESULT ((BASICSELECTOR|ADVANCEDSELECTOR), RESULT)>

<!ELEMENT RESULT EMPTY>

<!ATTLIST RESULT

ID CDATA #REQUIRED

STATUS CDATA #REQUIRED

NUMINV CDATA #REQUIRED

>
```

The body of a valid invalidation request should begin with

```
<?xml version="1.0" ?>

<!DOCTYPE INVALIDATION SYSTEM "internal:///WCSinvalidation.dtd">
```

Therefore the root element for an invalidation request must be `INVALIDATION`, which contains a list of one or more `OBJECT` elements.

And the body of the invalidation response begins with

```
<?xml version="1.0"?>

<!DOCTYPE INVALIDATIONRESULT SYSTEM "internal:///WCSinvalidation.dtd">
```

Likewise, the root element for an invalidation response must be `INVALIDATIONRESULT`, which contains a list of one or more `OBJECTRESULT` elements, each one of which matches one `OBJECT` element in the invalidation request.

Both `INVALIDATION` and `INVALIDATIONRESULT` take VESION as attribute to denote what version the `WCSinvalidation.dtd` is being used as XML document type.

Also note that the `SYSTEM` element is optional and intended for use to send system information by any implementation that supports `WCSinvalidation.dtd`. Each implementation can choose to use it or ignore it since it is optional.

The meanings for all the elements and attributes are discussed below.

Invalidation Selectors

An invalidation selector defines a set of page documents upon which the invalidation action is to be performed. There are two types of selectors: basic selectors and advanced selectors.

Basic. A basic selector contains an exact HTTP URI, which specifies what page document to be invalidated. The definitions for URI and its comparison are found in HTTP/1.1 specification. It is easy to see that a basic selector can select either none or exactly one page object. An implementation chooses to ignore the host name in URI and only keeps the *abs_path*. For definition of abs_path, also refer to HTTP/1.1 specification.

Advanced. An advanced selector is defined based on the following attributes and elements:

URIPREFIX (required)	URIPREFIX attribute specifies a URI path prefix and therefore it must begin and end with a slash '/'. Same as with basic URI, if host name is present in URIPREFIX, it is extracted (and may be ignored in some implementation). This required field specifies that in the set of page documents that the current selector defines, all documents have this common path prefix. Any subsequent attributes and elements are used in conjunction with URIPREFIX to further limit the matching of set of page documents. If no more element is specified, invalidation simply means the expiration of all page documents with the same path prefix.
URIEXP (optional)	URIEXP attribute is a regular expression and its scope is the *abs_path* part of the URI. Please refer to HTTP specification for more definitions. Therefore if the URI contains the following regular expression special characters literally, they need to be escaped. Reserved regular expression characters include period ".", question mark "?", asterisk "*", bar "|", parentheses "()", brackets "[]", curly braces "{}", carat "^", dollar sign "$", and backslash "\". Caveat: many people confuse Unix shell filename shorthand with regular expression. For example, when you type "`ls -l *.c`" under shell, the last part, "`*.c`" is really *not* regular expression. Please refer to regular expression specification for more. Since URIEXP field is used in conjunction with URIPREFIX, regular expression matching is done only to the page objects having the same URIPREFIX path prefix. Therefore users should be cautious not to supply a URIEXP field that is contradictory to the URIPREFIX field, especially when URIEXP contains the use of "^".

	Contradiction yields an empty set of page objects to be invalidated-which is always successful.
HOST (optional)	HOST attribute specifies host name. It should contain same information as in HTTP/1.1 host header. Or it can also be extracted from URI, URIPREFIX and/or URIEXP if any one of them contains host name. There should not be any conflict.
METHOD (optional)	METHOD attribute describes which HTTP method is used. Currently it is either GET or POST. And when this field is blank, it defaults to GET.
BODYEXP (optional)	BODYEXP attribute is the regular expression matching the HTTP request body. It is only meaningful when METHOD field is POST. Same as with URIEXP, BODYEXP field cannot contain the special regular expression characters literally-they have to be escaped. When specifying the using of "^", users also need to be careful not to define empty set of invalidation objects by accident.
COOKIE (optional)	COOKIE elements contains NAME and VALUE attributes to describe an HTTP cookie. One advanced selector can have zero or more cookies. When a cookie is given, its name cannot be blank. The use of *value* varies depending on what kind of cookie it is. If the cookie is a session cookie, whatever the *value* is given, it is ignored. Otherwise, *value* is used literally to match against the cookies that the page objects have. Only objects having the common URIPREFIX path prefix are examined with COOKIE field.

HEADER (optional)	HEADER element contains NAME and VALUE attributes to describe an HTTP/1.1 header. One advanced selector can have zero or more headers. When a header is given, its name cannot be blank. Only objects having the common URIPREFIX path prefix are examined with HEADER field.
OTHER (optional)	OTHER element is designed to support any other type of aspects for a page document other than COOKIE and HEADER. It has 3 attributes which define what TYPE, NAME and VALUE this aspect is/has. Each implementation has the freedom to utilize this element at their convenience.

The advanced selector is more sophisticated than the basic selector. Its descriptive capability in URLs is as powerful as regular expression itself. In fact, a basic selector can be expressed in the form of the advanced selector.

For example, suppose the URI in basic selector is "`/p1/p2/p3/file.htm`", then in the equivalent advanced selector, the URIPREFIX is "/p1/p2/p3/" and the URIEXP is "`^/p1/p2/p3/file.htm$`". If you choose to specify "/" for the URIPREFIX, it is still okay. Since the regular expression is done against the set of page objects containing the common URIPREFIX path prefix, it is obvious that the smaller the set, the more efficient the invalidation.

Invalidation Actions

An invalidation action consists of an implicit immediate expiration and an optional removal TTL (or time-to-live). The removal TTL is non-negative number in seconds. If the removal TTL is missing, then it means to remove the selected page documents on demand in the next appropriate cycle.

Since in one invalidation XML message there can be more than one invalidation object, it is possible that some page documents are affected by more than one action if they are selected by more than one invalidation objects. In this situation, the earliest removal time always prevails.

Invalidati n Results

An invalidation result contains 3 fields and they are all of string type.

ID	ID is a sequence number to disambiguate the selectors. Since it is legitimate but not common for invalidation objects in invalidation request to contain the same selector on purpose or by accident, ID field is used to tell them apart in the invalidation response.
STATUS	STATUS is a string to describe the status for the action given to Webcache to invalidate what selector selects. Examples of STATUS value can be "SUCCESS", "URI NOT FOUND", "URI NOT CACHEABLE", etc.
NUMINV	NUMINV is a number to describe how many page documents are successfully invalidated.

Invalidation Clients

Any client that is capable of sending and receiving HTTP messages can be used to do invalidation clients. The simplest client is Telnet. Here is a walk-through of the transcript for an invalidation request and response done by using Telnet program in one invalidation implementation.

```
1       POST /x-invalidate HTTP/1.0
2       Authorization: Basic aW52YWxpZGF0b3I6aW52YWxpZGF0b3I=
3       Content-Length: 217
4
5       <?xml version="1.0" ?>
6       <!DOCTYPE INVALIDATION SYSTEM "invalidation.dtd">
7       <INVALIDATION VERSION="WCS-1.0">
8       <OBJECT>
9       <BASICSELECTOR URI="/cache.htm" />
10      <ACTION />
11      </OBJECT>
12      </INVALIDATION>
```

Line 1 specifies using HTTP/1.0 POST method to request "`/x-invalidate`". All invalidation requests must contain a URI that the implementation chooses to use.

Line 2 specifies using HTTP basic authentication and "`aW52YWxpZGF0b3I6aW52YWxpZGF0b3I=`" really is "`invalidator:invalidator`" after the base64 encoding.

Line 3 specifies the content length.

Line 4 is the separator line.

Line 5 declares the following HTTP body contains an XML document.

Line 6 specifies the root element of the XML document to be `INVALIDATION` and its definition of document type is given by `WCSinvalidation.dtd`.

Lines 7 ~ 12 contains one invalidation object which intends to invalidate "`/cache.htm`" if it is in the cache.

Here is the response for a successful invalidation of "`/cache.htm`".

```
1       HTTP/1.1 200 OK
2       Date: Sun, 22 Apr 2001 07:54:09 GMT
3       Allow: GET, HEAD
4       Server: Webserver/2.0.0.0.0
5       Content-Type: text/html
6       Content-Length: 284
7
8       <?xml version="1.0"?>
9       <!DOCTYPE INVALIDATIONRESULT SYSTEM "invalidation.dtd">
10      <INVALIDATIONRESULT VERSION="WCS-1.0">
11      <OBJECTRESULT>
12      <BASICSELECTOR URI="/cache.htm" />
13      <RESULT ID="1" STATUS="SUCCESS " NUMINV="1"/>
14      </OBJECTRESULT>
15      </INVALIDATIONRESULT>
```

Lines 1 contains the `200` HTTP response code to denote successful invalidation.

Line 2 ~ 7 are other HTTP headers. Please refer to HTTP and ESI standards for more details.

Line 7 is the separator line between headers and body.

Line 8 declares that the HTTP body is an XML document.

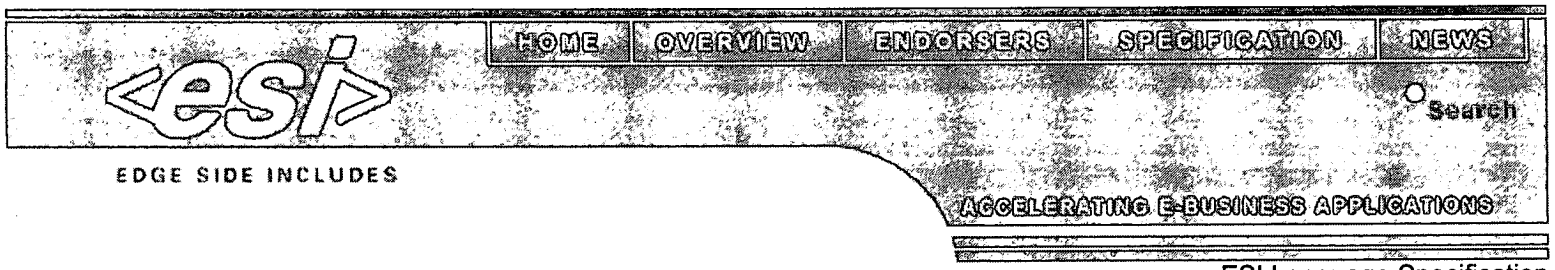
Line 9 specifies the root element of the XML document to be `INVALIDATIONRESULT` and its definition of document type is also given by `WCSinvalidation.dtd`.

Lines 10 ~ 15 contains one invalidation object result, which says the action specified against the selection is successful, and there is 1 match for the selection in the cache.

back to top

***

Home | Overview | Endorsers | Specification | News | Contact Us



ESI Language Specification
Edge Architecture Specification
ESI Invalidation Protocol
JESI Tag Library Specification

Technical Specification

Edge Architecture Specification

Authors:
Mark Nottingham, Akamai
Xiang Liu, Oracle

Table of Contents:

Abstract

This document defines the Edge Architecture, which extend the Web infrastructure through the use of HTTP surrogates - intermediaries that act on behalf of an origin server.

1. Introduction

One approach to scaling the Web is the use of *surrogates* - intermediaries that act on behalf of and with authority of an origin server (also known as "reverse proxies"). This document describes a framework for the distribution of HTTP content by surrogates, by specifying a means of controlling surrogates with HTTP headers, along with caching and response processing models for them.

Surrogates may be deployed close to the origin server, or throughout the network - a configuration often referred to as a "Content Delivery Network" (CDN).

Because they act on behalf of the origin server (and therefore the content's owner), surrogates offer content owners greater control over their behavior than proxies. As a result, they offer greater potential for improving performance, offloading processing from the origin server, and adding unique functionality to the Web.

This document uses the Extended BNF syntax and rules from HTTP/1.1 [HTTP].

2. Controlling Surrogates with HTTP Headers

Unlike standard HTTP intermediaries, Surrogates offer the ability for finer control by the origin server and content owner, because of their implied relationship. To enable this, we define HTTP headers to advertise the capabilities of a particular surrogate device, and control how it behaves.

2.1 Surrogate-Capability Header

The Surrogate-Capabilities request header allows surrogates to advertise their capabilities with *capability tokens*. Capability tokens indicate sets of operations (e.g., caching, processing) that a surrogate is willing to perform. They follow the form of product tokens in the HTTP;

capability = token ["/" token]

Capability tokens are case-sensitive.

As requests pass through surrogates, the Surrogate-Capabilities header is appended;

```
capability-hdr = "Surrogate-Capability" ":" 1#capability-set
capability-set = device-token "=" <"> capability \
                 *( SP capability ) <">
device-token = token
```

For example,

Surrogate-Capabilities: abc="Surrogate/1.0", def="Surrogate/1.0 ESI/1.0"

The name in each capability set identifies a *device token*, which uniquely identifies the surrogate that appended it. Device tokens must be unique within a request's Surrogate-Capabilities header.

The value contains a space-separated list of capability tokens. In the example above, two surrogates are present in the request chain; one identified as 'abc' is capable of applying "ESI/1.0", while 'def' is capable of handling both "Surrogate/1.0" and "ESI/1.0".

Surrogates must only append their information to any existing Surrogate-Capability headers, so that it may be read from left to right (or, if the headers are separate, from top to bottom) to construct a list of surrogates that the request passed through (and thus a list that may be read from right to left to discover surrogates that the response will pass through).

2.2 Surrogate-Control Header

The Surrogate-Control response header allows origin servers to dictate how surrogates should handle response entities, with *control directives*. Currently defined directives control processing and cache behavior.

```
control-hdr = "Surrogate-Control" ":" 1#( control-directive \
              *( ";" device-token ) )
control-directive = no-store |
                    no-store-remote |
                    max-age |
                    content |
                    extension-directive
no-store = "no-store"    ; see section 4.2.1
no-store-remote = "no-store-remote"    ; see section 4.2.2
max-age = "max-age" "=" \
          delta-seconds [ "+" delta-seconds ]   ; see section 4.2.3
content = "content" "=" \
          <"> capability *( SP capability ) <"> ; see section 4.2.4
extension-directive = token [ "=" ( token | quoted-string ) ]
```

For example,

Surrogate-Control: no-store, content="ESI/1.0"

Surrogates may modify the Surrogate-Control header to instruct downstream surrogates to process capabilities originally targetted at them. If no downstream surrogates have identified themselves, the header should be stripped from responses.

2.2.1 no-store

The no-store directive specifies that the response entity should not be stored in cache; it is only to be used for the original request, and may not be validated on the origin server.

2.2.2 no-st re-rem te

The no-store-remote directive has similar semantics to the no-store directive, except that it should only be honored by those surrogates that consider themselves "remote". Generally, this means those that are more than one or two hops from the origin server, such as surrogates in a CDN.

2.2.3 max-age

The max-age directive specifies how long the response entity can be considered fresh, in seconds. After this time, implementations must consider the cached entity stale.

For example,

max-age=30

Optionally, a '+' and a *freshness extension* can be appended, that specifies an additional period of time (in seconds) the stale entity may be served, before it must be revalidated or refetched as appropriate.

For example,

max-age=30+60

If no freshness extension is specified, it should be considered as '0' (i.e., the object should be revalidated or refetched immediately).

2.2.4 content

The content directive identifies what processing surrogates should perform on the response before forwarding it. The value of the content directive is a left-to-right ordered, space-separated list of capabilities for processing by surrogates.

For example,

content="ESI/1.0 ESI-Inline/1.0"

This directive specifies that first the operations represented by the "ESI/1.0" capability token and then the "ESI-Inline/1.0" capability token should be applied to the response entity. See also "Response Processing Model".

Once processing takes place, the capability token that invoked it (as well as the 'content' directive, if appropriate) is consumed; that is, it is not passed forward to surrogates.

2.3 Surrogate-Control Targetting

Because surrogates can be deployed hetrogeneously in a hierarchy, it is necessary to enable the targetting of directives at individual devices.

Surrogate-Control directives may have a parameter that identifies the surrogate that they are targetted at, as identified by the device token in the request's Surrogate-Capabilities header. Directives without targetting parameters are applied to all surrogates, unless a targetted directive overrides it. For example,

Surrogate-Control: max-age=60;abc, max-age=300

Here, the device that identified itself as 'abc' in the Surrogate-Capabilties request header will apply a max-age of 300; all other surrogates will apply a max-age of 60.

Surrogate-Control: content="ESI/1.0";abc, content="ESI-Inline/1.0";def

This header specifies that the device that identified itself as 'abc' should process the response entity for ESI, while the surrogate that identified itself as 'def' should process it for ESI-Inline.

Implementations are not required to support targetting.

3. Caching Model

Caching in surrogates operates in a manner similar to the HTTP; the same freshness and validation mechanisms form its basis. However, there are additional mechanisms for controlling cacheability in Surrogates, that override such mechanisms in the HTTP.

The Surrogate-Control response header contains several directives that influence entity cacheability; specifically, "no-store", "no-store-remote", and "max-age" (see "Surrogate-Control Header" for more information). Collectively, these directives and their behaviors are described by the capability token

Surrogate/1.0

This token should be included in all requests sent by compliant surrogates (see "Surrogate-Capabilities Header").

When any of these directives are present, they override any HTTP cacheability information present in the response. If more than one is targetted at a surrogate, the most specific applies. For example,

Surrogate-Control: max-age=60, no-store;abc

The surrogate that identified itself as 'abc' (see "Controlling Surrogates with HTTP Headers") would apply no-store; others would apply max-age=60. Conversely,

Surrogate-Control: no-store, max-age=60;abc

In this example, 'abc' would apply max-age=60, while other surrogates would apply no-store.

Surrogates should ignore any HTTP Cache-Control request header directives.

4. Response Processing Model

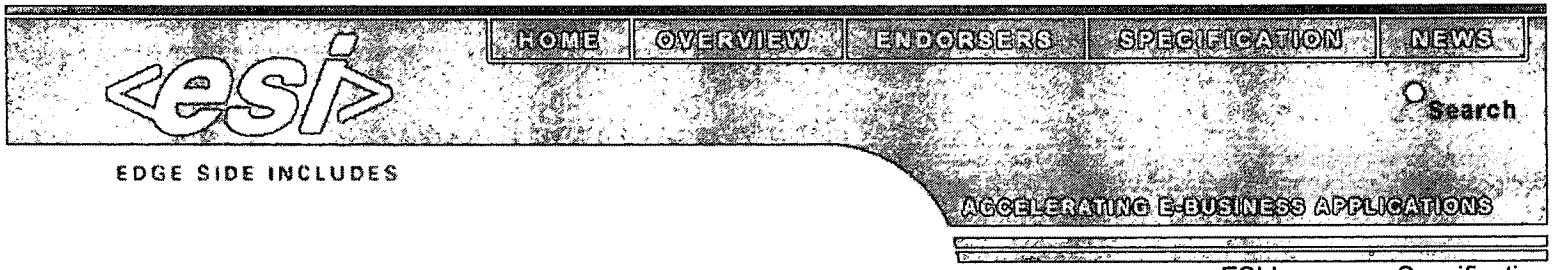
Surrogates may also invoke *processors* on response entitities as they pass through. Examples of processing include images transcoding, the application of XSLT stylesheets, or the interpretation of an in-markup language. By default, processing takes place after caching; that is, cached entities have any applicable processing applied before being served. Processors and extension directives may modify this behavior.

Processing is invoked with the 'content' surrogate-control directive.

back to top

***

Home | Overview | Endorsers | Specification | News | Contact Us



ESI Language Specification
Edge Architecture Specification
ESI Invalidation Protocol
JESI Tag Library Specification

Technical Specification

ESI Language Specification 1.0

Authors:
Mark Tsimelzon, Akamai
Bill Weihl, Akamai
Larry Jacobs, Oracle
Editors:
Mark Nottingham, Akamai

Table of Contents:

Abstract
This specification defines ESI 1.0, the Edge Side Includes language, which allows content assembly by HTTP surrogates, by providing an in-markup XML-based language.

1. Introduction

Edge Side Includes (ESI) is an XML-based markup language that provides a means to assemble resources in HTTP clients. Unlike other in-markup languages, ESI is designed to leverage client tools like caches to improve end-user perceived performance, reduce processing overhead on the origin server, and enhanced availability. ESI allows for dynamic content assembly at the edge of the network, whether it is in a Content Delivery Network, end-user's browser, or in a "Reverse Proxy" right next to the origin server.

ESI is primarily intended for processing on surrogates (intermediaries that operate on behalf of the origin server, also known as "Reverse Proxies") that understand the ESI language. However, its application is not restricted to these devices. The control of where ESI is processed is addressed in [Edge Architecture]. Its capability token is

ESI/1.0

ESI allows surrogates to treat parts of pages as cacheable resources, which gives them the ability to serve resources from cache in more situations.

1.1 Relationship to Other Standards

ESI is an XML language designed to be interposed into markup to provide logic and dispatch services, targetted for processing after the markup has left the origin server, but before it is paginated by the end user's client. As a result, the markup that is emitted by the origin server is not valid; it contains interposed elements from the ESI namespace. Additionally, it may contain native markup (e.g., HTML) that will not exist in the paginated entity.

XInclude is a W3C effort to standardize a general inclusion mechanism for XML. The inclusion aspect of ESI is somewhat similar to XInclude, with additional semantics for failure handling. Additionally, ESI processing is targetted (usually, to surrogates), while XInclude doesn't define explicit or implicit targetting of processing. It is our hope that future development of ESI and XInclude may be coordinated, bringing them into alignment.

Additionally, ESI borrows some portions of the XSLT language for logic and processing control.

1.2 Relationship to in-Markup Content Generation and Client Scripting Languages

There are several proprietary in-markup languages available (e.g., PHP, SSI, etc.), as well as a few standardized solutions (e.g., EMCAScript, etc.). ESI is not intended to replace these languages. It is expected that ESI will be used in concert with both content generation and client scripting markup.

When ESI is processed on the same device as other markup (e.g., Server Side Includes), some form of precedence in processing will need to be defined. This determination is currently implementation-specific.

2. ESI Functional Overview

The ESI language is conceptually similar in many ways to the Server Side Includes (SSI) function found in many web servers. It is an in-markup scripting language that is interpreted before the page is served to the client.

Version 1.0 includes the following functionality:

- **Inclusion** - ESI can compose pages by assembling included content, which is fetched from the network. This allows each such *fragment* to have its own metadata (e.g., cacheability and handling information) seperately associated.
- **Variable support** - ESI 1.0 supports the use of variables based on HTTP request attributes in a manner reminiscent of the Common Gateway Interface. These variables can be used by ESI statements or written directly into the processed markup.
- **Conditional processing** - ESI allows conditional logic with Boolean comparisons to be used to influence how a template is processed.
- **Exception and error handling** - ESI provides for specification of alternate and default resources in a number of situations.

The ESI assembly model is comprised of a *template* containing *fragments*. The template is the container for assembly, with instructions for the retrival of fragments, and is the resource associated with the URL the end user requests. It includes ESI elements that instructs *ESI Processors* (clients that understand ESI) to fetch and include a fragment's URI. The fragments themselves can be any textual Web resource, typically HTML markup.

Because fragments are separate resources, they can be assigned their own cacheability and handling information. For example, a cache time-to-live (TTL) of several days could be appropriate for the template, but a fragment containing a frequently-changing story or ad may require a much lower TTL. Some fragments may require being marked uncacheable.

3. ESI Elements

ESI elements are XML, in an ESI-specific XML Namespace. This allows them to be embedded in many common Web document formats, including HTML and XML-based server-side processing languages. ESI Processors parse but do not process elements outside of the ESI namespace. When an ESI Processor processes a template, ESI elements are

stripped from the output.

The XML Namespace for ESI 1.0 is

http://www.edge-delivery.org/esi/1.0

Future versions of and extensions to ESI will use distinct namespaces. Typically, documents will declare the ESI namespace in the top-level element; for templates, this would be the <html> tag, while in fragments this could be a tag wrapping the entire fragment.

ESI element names and attribute names are always lowercase.

Some ESI elements may contain arbitrary markup, such as HTML or other XML; others may not. See the elements definitions for details on the legal contents of a particular element. For example, the **strong** lines below are invalid and will be discarded by the ESI Processor.

```
<esi:try>
  Invalid markup here
    <esi:attempt>
        <esi:include ... >
        This line is valid and will be processed.
    </esi:attempt>
        Invalid markup here
    <esi:except>
        This HTML line is valid and will be processed.
    </esi:except>
    Invalid markup here
</esi:try>
```

3.1 include

The `include` element specifies a fragment for assembly and allows for two optionally specified behaviors. `include` is an empty element; it does not have a closing tag.

<esi:include src="URI" alt="URI" onerror="continue" />

For example,

<esi:include src="http://example.com/1.html" alt="http://bak.example.com/2.html" onerror="continue"/>

<esi:include src="http://example.com/ search?query=$(QUERY_STRING{query})"/>

The `include` statement tells ESI Processors to fetch the resource specified by the `src` attribute. This can be an simple URI, as shown in the first example, or can include variables (see "Variables"), as shown in the second. In either case, the final

attribute value must be a valid URI. Relative URIs will be resolved relative to the template. The resulting object will replace the element in the markup served to the client.

ESI Processor implementations may limit the number of `includes` used in a single ESI resource. Additionally, they may limit the number and/or depth of included documents that will be recursed. This assures that ESI processing does not monopolize resources or impact end-user perceived performance.

The optional `alt` attribute specifies an alternative resource if the `src` is not found. The requirements for the value are the same as those for `src`. Some ESI Processors may not implement this attribute, depending on its applicability; for example, surrogates near the origin server typically cannot usefully process them.

If an ESI Processor can fetch neither the `src` nor the `alt`, it returns a HTTP status code greater than 400 with an error message, unless the `onerror` attribute is present. If it is, and `onerror="continue"` is specified, ESI Processors will delete the `include` element silently.

3.2 inline

ESI fragments need not be fetched independently by the ESI processor. The inline element provides a way to demarcate fragments, embedded in the HTTP response. These fragments are stored and assembled in the ESI processor as independently included fragments are handled. Inline has a closing tag.

```
<esi:inline name="URI" fetchable="{yes | no}">
    fragment to be stored within an ESI processor
</esi:inline>
```

The inline statement is used to demarcate ESI fragments. The fragment is embedded within an HTTP response to an ESI processor. The ESI processor will parse response and extract all inline fragments and store them independently, under the URI specified.

Some inline fragments are only delivered as part of an HTTP response for another object. These are said to be not independently fetchable by the ESI processor. When a non fetchable fragment is needed by the ESI processor, the ESI processor must request the object from which the inline fragment was extracted.

An independently fetchable fragment may be requested by the ESI processor by using its name as the URI.

Implementation of inline is optional; ESI Processors use the capability token

ESI-Inline/1.0

to advertise their willingness to process this tag.

3.3 ch se | when | otherwise

These conditional elements add the ability to perform logic based on expressions. All three must have an end tag.

```
<esi:choose>
    <esi:when test="...">
        ...
    </esi:when>
    <esi:when test="...">
        ...
    </esi:when>
    <esi:otherwise>
        ...
    </esi:otherwise>
</esi:choose>
```

Every `choose` element must contain at least one `when` element, and may optionally contain exactly one `otherwise` element. No other ESI elements or non-ESI markup can be direct children of a `choose` element.

ESI processors will execute the first `when` statement whose `test` attribute evaluates truthfully, and then exit the `choose` element. If no `when` element evaluates to true, and an `otherwise` element is present, that element's content will be executed. See "ESI Expressions" for the syntax of the `test` attribute's value.

ESI elements as well as non-ESI markup can be included inside `when` or `otherwise` elements.

For example:

```
<esi:choose>
    <esi:when test="$(HTTP_COOKIE{group})=='Advanced'">
        <esi:include src="http://www.example.com/advanced.html"/>
    </esi:when>
    <esi:when test="$(HTTP_COOKIE{group})=='Basic User'">
        <esi:include src="http://www.example.com/basic.html"/>
    </esi:when>
    <esi:otherwise>
        <esi:include src="http://www.example.com/new_user.html"/>
    </esi:otherwise>
</esi:choose>
```

3.4 try | attempt | except

Exception handling is provided by the `try` element, which must contain exactly one instance of both an `attempt` and an `except` element (all with end tags):

```
<esi:try>
    <esi:attempt>
        ...
    </esi:attempt>
    <esi:except>
        ...
    </esi:except>
```

```
</esi:try>
```

Valid children of `try` are `attempt` and `except`; no other ESI or non-ESI markup can be a direct child. `attempt` and `except` may contain valid ESI or non-ESI markup.

ESI Processors first execute the contents of `attempt`. A failed ESI `include` statement will trigger an error and cause the ESI Processor to execute the contents of the `except` element. Statements other than `include` and `inline` will not trigger this error.

In this example, the `attempt` is to fetch an ad. If the ad fetch fails, a static link will be included instead.

```
<esi:try>
    <esi:attempt>
        <esi:comment text="Include an ad"/>
        <esi:include src="http://www.example.com/ad1.html"/>
    </esi:attempt>
    <esi:except>
        <esi:comment text="Just write some HTML instead"/>
        <a href=www.akamai.com>www.example.com</a>
    </esi:except>
</esi:try>
```

3.5 comment

The `comment` element allows developers to comment their ESI instructions, without making the comments available in the processor's output. `comment` is an empty element, and must not have an end tag.

<esi:comment text="..." />

For example:

<esi:comment text="the following animation will have a 24 hr TTL." />

Comments are not evaluated by ESI Processors; they are deleted before output. Comments that need to be visible in the output should use standard XML/HTML comment syntax.

3.6 rem ve

The `remove` element allows for specification of non-ESI markup for output if ESI processing is not enabled.

<esi:remove> ... </esi:remove>

For example:

```
<esi:include src="http://www.example.com/ad.html"/>
<esi:remove>
  <a href="http://www.example.com">www.example.com</a>
</esi:remove>
```

Normally, when this block is procesed, the ESI Processor fetches the ad.html resource and includes it in the template while silently discarding the `remove` element and its contents.

If for some reason ESI processing is not enabled, all of the elements will be passed through to clients, which will ignore markup it doesn't understand.

With Web clients, this works because browsers ignore invalid HTML, such as `<esi:...>` and`</esi:...>` elements, leaving the HTML `a` element and its content.

The `remove` statement cannot include nested ESI markup.

3.7 vars

To include an ESI variable in markup outside an ESI block, use the `vars` element.

<esi:vars> ... </esi:vars>

For example:

```
<esi:vars>
  <img src="http://www.example.com/$(HTTP_COOKIE{type})/hello.gif"/ >
</esi:vars>
```

See "ESI Variables" for more information about variables.

3.8 <!--esi ...-->

This is a special construct to allow HTML marked up with ESI to render without processing. ESI Processors will remove the start ("<!--esi") and end ("-->") when the page is processed, while still processing the contents. If the page is not processed, it will remain, becoming an HTML/XML comment tag. For example,

```
<!--esi
  <p><esi:vars>Hello, $(HTTP_COOKIE{name})!</esi:vars></p>
-->
```

This assures that the ESI markup will not interfere with the rendering of the final HTML if not processed.

4. ESI Variables

ESI 1.0 supports the following read-only variables, which are based on the client's HTTP request line and headers:

ESI Variables

Variable Name	HTTP Header	Substructure Type	Example
HTTP_ACCEPT_LANGUAGE	Accept-Language	list	da, en-gb, en
HTTP_COOKIE	Cookie	dictionary	id=571; visits=42
HTTP_HOST	Host	-	esi.xyz.com
HTTP_REFERER	Referer	-	http://roberts.xyz.com/
HTTP_USER_AGENT	User-Agent	dictionary (special)	Mozilla; MSIE 5.5
QUERY_STRING	-	dictionary	first=Robin&last=Roberts

Variable names are always uppercase. To reference a variable, surround the name with parenthesis and append a dollar sign ($).

For example:

$(HTTP_HOST)

See "Expressions and Operators" and the vars element for information on the use of variables in ESI markup.

4.1 Variable Substructure Access

By default, ESI variables are evaluated in a string context. However, some which represent more complex data will make automatically parsed and typed data available.

To access a variable's substructure, the variable name should be appended with braces containing the key which is being accessed. For example,

$(HTTP_COOKIE{username})

Variables capable of subaccess can be classified as dictionairies or lists, as outlined in the "ESI Variables" table.

4.1.1 Dictionary

Variables identified as dictionaries make access to strings available through their appropriate keys. Dictionary keys are case-sensitive. the For example,

$(HTTP_COOKIE{visits})

evaluates to "42" if the Cookie header contains the string "visits=42".

The dictionary associated with the User-Agent header contains only three special values; browser, version and os.

browser allows access to the ESI Processor's determination of the User-Agent's browser, and may be either "MOZILLA", "MSIE", or "OTHER".

version allows access to the User-Agent's browser version.

os allows access to the Processor's determination of the User-Agent's platform, and may evaluate to either "WIN", "MAC", "UNIX" or "OTHER".

4.1.2 List

Variables identified as lists will return a boolean value depending on whether the requested value is present. For example,

$(HTTP_ACCEPT_LANGUAGE{en-gb})

will return true if the string "en-gb" is in the appropriate header; otherwise, it will return false.

4.2 Variable Default Values

Variables whose values are empty, nonexistent variables and undefined substructures of variables will evaluate to an empty string when they are accessed. The logical or operator can be used to specify a default value in an expression where desirable, in the form:

$(VARIABLE|default)

For example:

<esi:include src="http://example.com/$(HTTP_COOKIE{id}|default).html"/>

will result in the ESI Processor fetching the following URI if the cookie "id" is not in the request.

http://example.com/default.html

As with other literals, if whitespace needs to be specified, the default value must be single-quoted.

$(HTTP_COOKIE{first_name}|'new user')

5. ESI Expressions

Conditional elements use expressions (in their test attributes) to determine how to apply the contained elements. Expressions consist of operators, variables and literals, and evaluate to true or false.

Single quotes are used within an expression to delimit literals. For example,

$(HTTP_HOST) == 'example.com'

Whitespace is optional around all operators, except `has`, which must be surrounded by whitespace.

5.1 Operators

The following set of unary and binary logical operators are supported by ESI expressions, listed in order of decreasing precedence:

Operators

Operator	Type
==, !=, <, >, <=, >=	comparison
!	unary negation
&	logical and
|	logical or

Operands associate from left to right. Sub-expressions can be grouped with parentheses in order to explicitly specify association.

If both operands are numeric, the expression is evaluated numerically. If either binary operand is non-numeric, both operands are evaluated as strings. After expansion, variables are loosely typed, but care should be taken. For example, a version reported as 3.01.23 or 1.05a will not test as a number.

The behavior of comparisons which incompatibly typed operators is undefined. If an operand is empty or undefined, the expression will always evaluate to false.

Logical operators ("&", "|", "!") can be used to qualify expressions, but cannot be used as comparitors themselves.

For example, the following expressions are correct:

```
!(1==1)
!('a'<='c')
(1==1)|('abc'=='def')
(4!=5)&(4==5)
```

while these will not evaluate:

```
(1 & 4)
("abc" | "edf")
```

6. Protocol Considerations

When an ESI template is processed, a separate request will need to be made for each include encountered. Implementations may use the original request's headers (e.g., Cookie, User-Agent, etc.) when doing so. Additionally, response headers from fragments (e.g., Set-Cookie, Server, Cache-Control, Last-Modified) may be ignored, and should not influence the assembled page.

Appendix A: Acknowledgements

The Authors would like to thank the following for the early and continuing development of the ESI language: Sriram Sankar, Andy Davis, Sam Gendler, Joshua Silver, Jay Parikh, Inbar Zamir, Ziv Katzir, and Chris Weikart.

back to top

***

Home | Overview | Endorsers | Specification | News | Contact Us